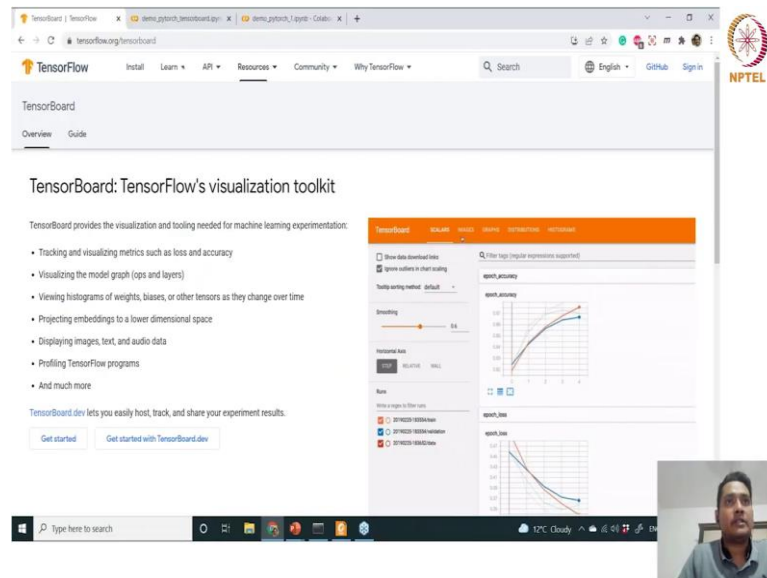


**Applied Accelerated Artificial Intelligence**  
**Dr. Satyajit Das**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Palakkad**

**Lecture - 21**  
**Introduction to PyTorch Part - 4**

(Refer Slide Time: 00:15)



(Refer Slide Time: 00:17)

The screenshot shows a Jupyter Notebook titled "demo\_pytorch\_tensorboard.ipynb". The code is as follows:

```
import torch
import torch.nn as nn
import torch.optim as opt
torch.set_printoptions(linewidth=120)
import torch.nn.functional as F
import torchvision
import torchvision.transforms as transforms
from torch.utils.tensorboard import SummaryWriter
from itertools import product

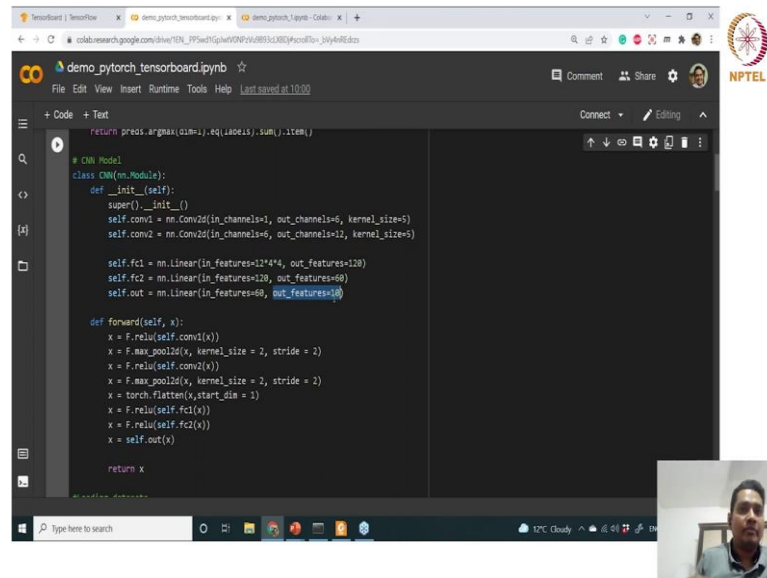
# helper functions
def get_num_correct(preds, labels):
    return preds.argmax(dim=-1).eq(labels).sum().item()

# CNN Model
class CNN(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, channels=6, kernel_size=5)
        self.conv2 = nn.Conv2d(6, channels=16, kernel_size=5)
```

So, let us go through this demo pytorch tensor board. So, here for tensorboard you use you need to use this tensorboard which is torch.utils.tensorboard toolkit ok. Now all

these imports that you have seen so far ok and these are again some a functional from torch and torch vision. So, these are the two important packages that we will use in this section of demo.

(Refer Slide Time: 00:52)



```
return pred.argmax(dim=1).eq(labels).sum().item()

# CNN Model
class CNN(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=6, kernel_size=5)
        self.conv2 = nn.Conv2d(in_channels=6, out_channels=16, kernel_size=5)
        self.fc1 = nn.Linear(in_features=12*4*4, out_features=120)
        self.fc2 = nn.Linear(in_features=120, out_features=60)
        self.out = nn.Linear(in_features=60, out_features=10)

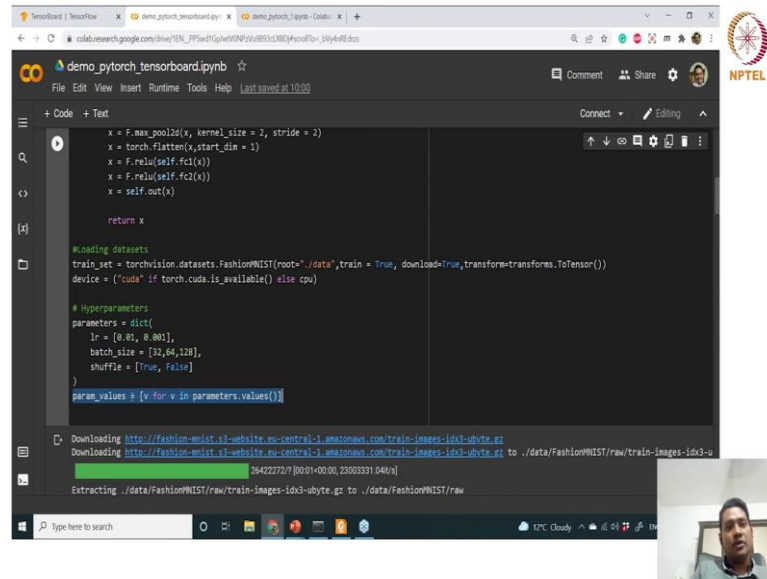
    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.max_pool2d(x, kernel_size=2, stride=2)
        x = F.relu(self.conv2(x))
        x = F.max_pool2d(x, kernel_size=2, stride=2)
        x = torch.flatten(x, start_dim=1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.out(x)

        return x
```

Now, the helper function this is doing nothing, but predicting the max value ok. So, that we will use. Now the CNN model is very very simple and you have seen such CNN model previously convolution1 convolution2 and fully connected 1 2 and last one which is also one fully connected layer and giving out features as 10 which is for your cifar 10 data set we are using that same training module ok.

So; that means, the 10 output feature it will actually predict and among them for which class it is giving your maximum prediction that will be your predicted points ok. So, we have defined the forward function as you have seen in so far in this previous demo and we are returning the x now this x is actually not the prediction, but output of the network ok.

(Refer Slide Time: 01:48)



```
from torch.nn import Conv2d, MaxPool2d, Flatten, Linear, ReLU, Softmax

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = Conv2d(1, 32, kernel_size=5)
        self.conv2 = Conv2d(32, 64, kernel_size=5)
        self.pool = MaxPool2d(2)
        self.fc1 = Linear(64 * 4 * 4, 100)
        self.fc2 = Linear(100, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.pool(x)
        x = self.fc1(x)
        x = self.fc2(x)
        return x

# Loading datasets
train_set = torchvision.datasets.FashionMNIST(root='./data', train=True, download=True, transform=transforms.ToTensor())
device = ("cuda" if torch.cuda.is_available() else "cpu")

# Hyperparameters
parameters = dict(
    lr=[0.01, 0.001],
    batch_size=[32, 64, 128],
    shuffle=[True, False]
)
param_values = [v for v in parameters.values()]

# Downloading data
train_loader = torch.utils.data.DataLoader(train_set, batch_size=32, shuffle=True)
optimizer = optim.Adam(model.parameters())
trainer = GradientDescentTrainer(model=model, data_loader=train_loader, optimizer=optimizer, device=device)
trainer.train()
```

So, when you are applying the cross entropy loss onto your output of this network output, then one softmax activation will be actually added. So, you do not need to add any other activations after the last layer it will automatically be added inside your cross entropy loss or particular loss you are using depending on that loss that PyTorch will decide which activations to use you know.

And you also can define in the in defining while defining the losses ok which activations treatments. Now we are downloading the data set defining the device ok and again. So, this is just the device which we will take this is just one for one GPU we are talking about for now and then the hyper parameters.

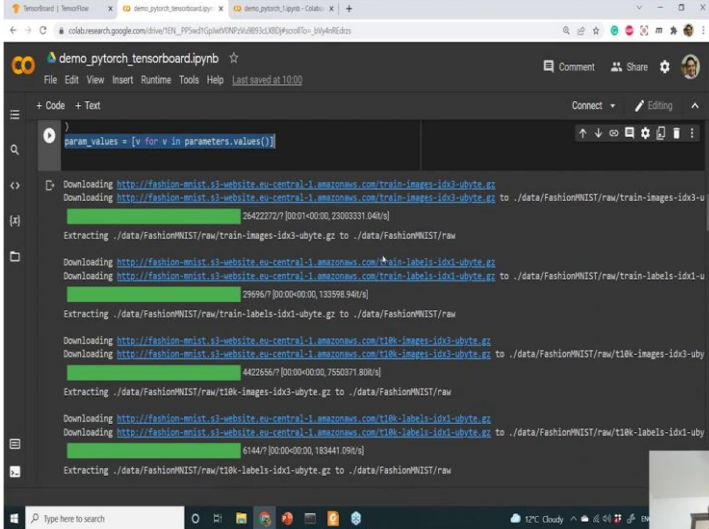
So, as I was mentioning that tensorboard that we use for see or tune the network parameters ok. So, or hyper parameters which will improve the training of or training outcome or prediction of your neural network. Now here you can see what are the hyper parameters we are using. We are using learning rate. So, one range we have defined.

So, basically we want to see among these range which learning rate will give me better result ok. So, this is just some definition we are using you can use your own definition for initializing this value. So, here we are actually getting this inside this list of parameters which is kind of a dictionary where we have this key and value pairs. So, key value key is the hyper parameter name itself and the values of the tuples for these things.

Now, batch size for again if you remember we have talked about different batch size will have different impact on the training outcome as well as the performance ok. So, for different batch size what kind of outcome we are getting that we want to see ok and then for the shuffle True and False. So, for training we mention before when we are loading the training data set ok. So, training data set we mentioned that training data set you need to use a shuffle equal to True.

But here we are actually explicitly saying that for both we want to see for True and False we want to see what kind of performance we will get because that way you will be satisfied that yes this is these are the parameters that can give me better results right. Now parameter values we are defining the parameter values in this taking all the values and storing in the parameter values. So, this we will use for training loop ok.

(Refer Slide Time: 05:08)



```

param_values = [v for v in parameters.values()]

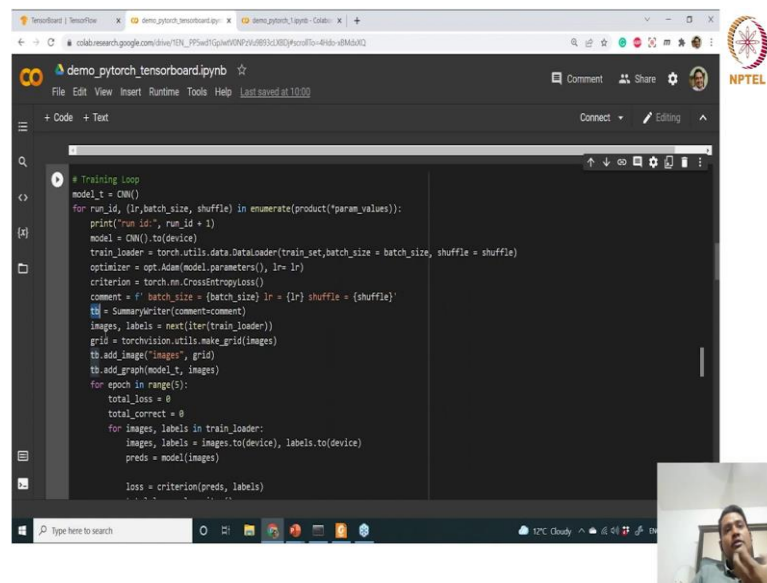
Downloading http://fashion-mnist.s3-website-eu-central-1.amazonaws.com/train-images-idx3-ubyte.gz
2542272/ [00:01<00:00, 2300331.04B/s]
Extracting ./data/FashionMNIST/raw/train-images-idx3-ubyte.gz to ./data/FashionMNIST/raw
Downloading http://fashion-mnist.s3-website-eu-central-1.amazonaws.com/train-images-idx3-ubyte.gz
2542272/ [00:01<00:00, 2300331.04B/s]
Extracting ./data/FashionMNIST/raw/train-images-idx3-ubyte.gz to ./data/FashionMNIST/raw
Downloading http://fashion-mnist.s3-website-eu-central-1.amazonaws.com/train-labels-idx1-ubyte.gz
299696/ [00:00<00:00, 133598.94B/s]
Extracting ./data/FashionMNIST/raw/train-labels-idx1-ubyte.gz to ./data/FashionMNIST/raw
Downloading http://fashion-mnist.s3-website-eu-central-1.amazonaws.com/t10k-images-idx3-ubyte.gz
442266/ [00:00<00:00, 7560371.80B/s]
Extracting ./data/FashionMNIST/raw/t10k-images-idx3-ubyte.gz to ./data/FashionMNIST/raw
Downloading http://fashion-mnist.s3-website-eu-central-1.amazonaws.com/t10k-labels-idx1-ubyte.gz
61447/ [00:00<00:00, 183441.09B/s]
Extracting ./data/FashionMNIST/raw/t10k-labels-idx1-ubyte.gz to ./data/FashionMNIST/raw

```

So, for each combination of this ok. So, for this lr for this batch size for this shuffle for this lr for this batch size, for this shuffling what is the outcome ok. So, for all these combinations we will see what is the outcome.

So, now we are downloading the data set. So, we are actually using FashionMNIST data set not cifar ten data set. So, this is also another data set. So, you can see this is also available in torchvision. So, torchvision.datasets or dot if you put dot you can see what are the data sets available we are using FashionMNIST dataset. So, all the FashionMNIST data set that we have downloaded.

(Refer Slide Time: 05:57)



```
# Training loop
model_t = CNN()
for run_id, (lr, batch_size, shuffle) in enumerate(product("param_values")):
    print("run id:", run_id + 1)
    model = CNN().to(device)
    train_loader = torch.utils.data.DataLoader(train_set, batch_size = batch_size, shuffle = shuffle)
    optimizer = opt.Adam(model.parameters()), lr= lr
    criterion = torch.nn.CrossEntropyLoss()
    comment = f' batch_size = {batch_size} lr = {lr} shuffle = {shuffle}'
    tb = SummaryWriter(comment=comment)
    images, labels = next(iter(train_loader))
    grid = torchvision.utils.make_grid(images)
    tb.add_image("images", grid)
    tb.add_graph(model_t, images)
    for epoch in range(5):
        total_loss = 0
        total_correct = 0
        for images, labels in train_loader:
            images, labels = images.to(device), labels.to(device)
            preds = model(images)
            loss = criterion(preds, labels)
```

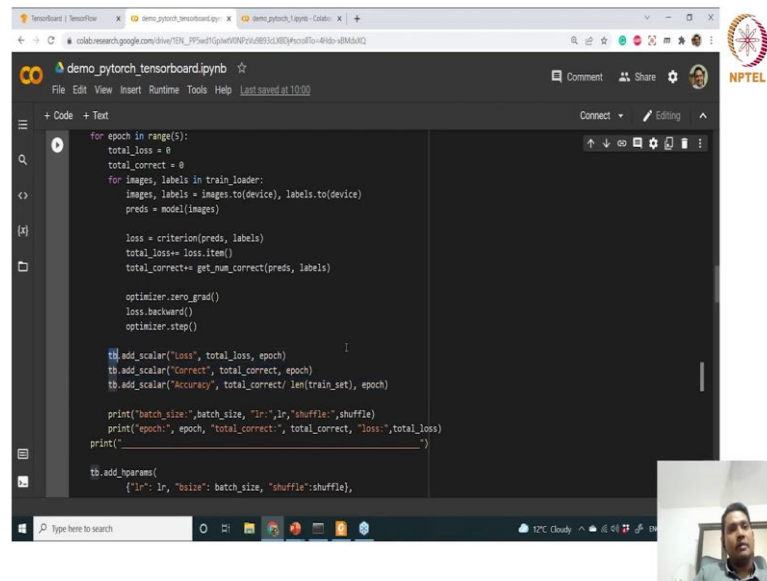
And then in the training loop we are defining the loop for the all the combinations now right.

So, now for run id. So, the for which run id. So, basically for each run it will influence. Now for each lr ok for this from this param values that we have defined previously well. So, for this lr for the each batch size for shuffle ok. So, what will be the performance? So, this model is defined to device. So, it will the CNN will get transferred to the device which is the gpu here then train loader will now training loading the train loader using the shuffle value either True or False depending on the value we are getting from param values.

Now, the criterion and optimizer that we have defined comment just for giving output like what kind of batch size and shuffle we are doing test bench summary now tb is essentially the SummaryWriter which is the tensorboard writer we are defining. So, SummaryWriter we have imported the tensorboard board as ok.

So, from the tensorboard we have imported this SummaryWriter a package which will actually give you the interface to write the different images different scalar values different graphs ok. So, tb is the object here now we are adding the images add images add graph function will add the graph. So, which add graph the model and with the images ok.

(Refer Slide Time: 07:48)

The image shows a Jupyter Notebook interface with a dark theme. The code is written in Python and implements a training loop for 5 epochs. It includes a data loader, forward pass, loss calculation, backward pass, and parameter updates. TensorBoard is integrated using the 'tb' module to log 'loss', 'correct', and 'accuracy' as scalars. The code also prints batch size, learning rate, and shuffle status. A small video inset in the bottom right corner shows a person's face. The browser address bar at the top indicates the file is hosted on Google Drive.

```
for epoch in range(5):
    total_loss = 0
    total_correct = 0
    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)
        preds = model(images)

        loss = criterion(preds, labels)
        total_loss += loss.item()
        total_correct += get_num_correct(preds, labels)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    tb.add_scalar("loss", total_loss, epoch)
    tb.add_scalar("correct", total_correct, epoch)
    tb.add_scalar("accuracy", total_correct / len(train_set), epoch)

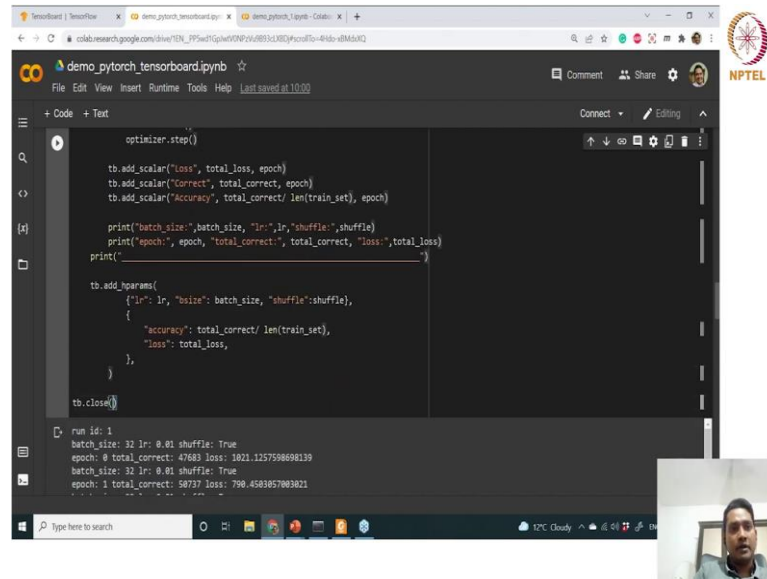
    print("batch_size:", batch_size, "lr:", lr, "shuffle:", shuffle)
    print("epoch:", epoch, "total_correct:", total_correct, "loss:", total_loss)
    print(" ")

    tb.add_params(
        {"lr": lr, "batch_size": batch_size, "shuffle": shuffle},
```

Now, we are starting the epoch. So, this for loop is essentially inside this for loop. So, for each combination we are now running this training right. So, train loss is 0 correct 0 now we have for each images batch size of images we are doing the forward pass then optimizing and doing the backward pass updating the parameters and after all the batches are processed in the forward once then we are actually adding what kind of loss we have gone with this.

So, now adding scalar will give you the. So, for each epoch. So, basically we are trying to plot this inside one line graph. So, that scalar will add to the scalar tab inside your tensorboard with all these parameter graphs. So, total loss versus epoch total correct versus epoch. So, with each epoch how the loss is improving with each epoch how the accuracy is improving ok correctness accuracy all these we are calculating and updating inside the tensorboard which is the tb.

(Refer Slide Time: 08:57)



The screenshot shows a Jupyter Notebook titled 'demo\_pytorch\_tensorboard.ipynb'. The code in the cell includes:

```
optimizer.step()

tb.add_scalar("Loss", total_loss, epoch)
tb.add_scalar("Correct", total_correct, epoch)
tb.add_scalar("Accuracy", total_correct / len(train_set), epoch)

print("batch_size:", batch_size, "lr:", lr, "shuffle:", shuffle)
print("epoch:", epoch, "total_correct:", total_correct, "loss:", total_loss)
print()

tb.add_params(
    {
        "lr": lr,
        "batch_size": batch_size,
        "shuffle": shuffle,
        "accuracy": total_correct / len(train_set),
        "loss": total_loss,
    },
)

tb.close()
```

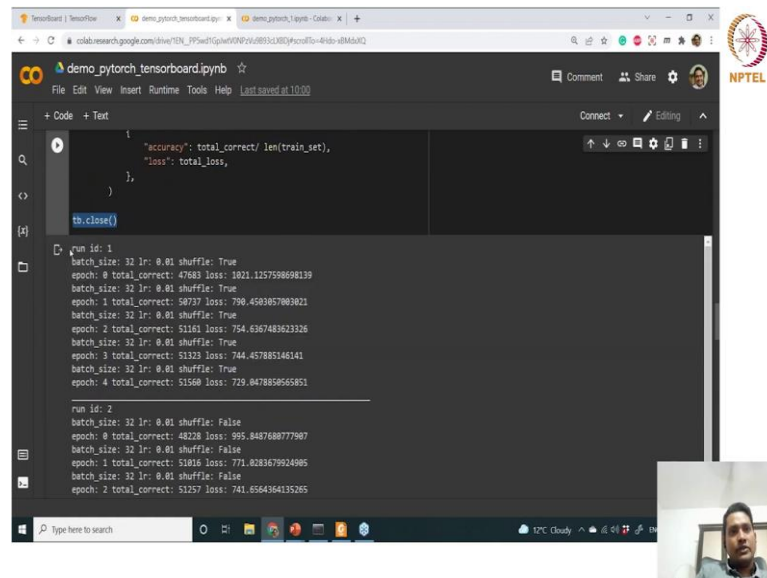
The output of the code shows the following data:

```
run id: 1
batch_size: 32 lr: 0.01 shuffle: True
epoch: 0 total_correct: 47683 loss: 1821.125758898139
batch_size: 32 lr: 0.01 shuffle: True
epoch: 1 total_correct: 50737 loss: 790.4503857003021
```

A small video inset in the bottom right corner shows a person speaking.

Then we are printing the batch size shuffle value the lr value which is the learning rate in the stochastic gradient descent optimizer and then just adding the hyper parameters to operate this. So, basically after running the entire loop we will just close this tb which is the tensor board.

(Refer Slide Time: 09:23)



The screenshot shows the same Jupyter Notebook as before, but with the code cell scrolled down to show the 'tb.close()' line. The output now includes data for epoch 2 and epoch 3:

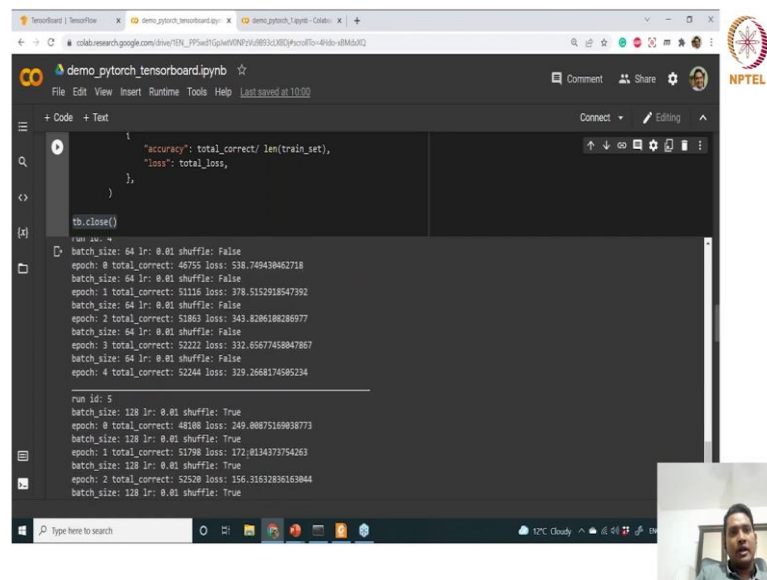
```
run id: 1
batch_size: 32 lr: 0.01 shuffle: True
epoch: 0 total_correct: 47683 loss: 1821.125758898139
batch_size: 32 lr: 0.01 shuffle: True
epoch: 1 total_correct: 50737 loss: 790.4503857003021
batch_size: 32 lr: 0.01 shuffle: True
epoch: 2 total_correct: 51161 loss: 754.636748363326
batch_size: 32 lr: 0.01 shuffle: True
epoch: 3 total_correct: 51323 loss: 744.457885140141
batch_size: 32 lr: 0.01 shuffle: True
epoch: 4 total_correct: 51560 loss: 729.0478850565851

run id: 2
batch_size: 32 lr: 0.01 shuffle: False
epoch: 0 total_correct: 48228 loss: 995.8487688777907
batch_size: 32 lr: 0.01 shuffle: False
epoch: 1 total_correct: 51010 loss: 773.8283679924095
batch_size: 32 lr: 0.01 shuffle: False
epoch: 2 total_correct: 51257 loss: 741.6584364135265
```

A small video inset in the bottom right corner shows a person speaking.



(Refer Slide Time: 09:25)

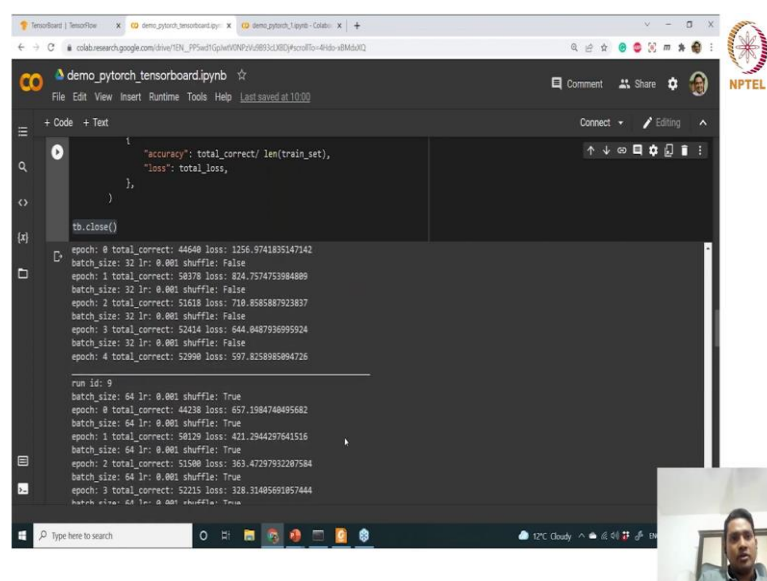


```
1 "accuracy": total_correct/ len(train_set),
2 "loss": total_loss,
3 },
4 )
5
6 tb.close()
```

batch\_size: 64 lr: 0.01 shuffle: False  
epoch: 0 total\_correct: 46755 loss: 538.749439462718  
batch\_size: 64 lr: 0.01 shuffle: False  
epoch: 1 total\_correct: 51816 loss: 378.5152918547392  
batch\_size: 64 lr: 0.01 shuffle: False  
epoch: 2 total\_correct: 51863 loss: 343.8386188286977  
batch\_size: 64 lr: 0.01 shuffle: False  
epoch: 3 total\_correct: 52222 loss: 332.65677458047867  
batch\_size: 64 lr: 0.01 shuffle: False  
epoch: 4 total\_correct: 52444 loss: 329.2668174985234

run id: 5  
batch\_size: 128 lr: 0.01 shuffle: True  
epoch: 0 total\_correct: 48188 loss: 249.80875169838773  
batch\_size: 128 lr: 0.01 shuffle: True  
epoch: 1 total\_correct: 51790 loss: 172.8134373794263  
batch\_size: 128 lr: 0.01 shuffle: True  
epoch: 2 total\_correct: 52520 loss: 156.31632836163844  
batch\_size: 128 lr: 0.01 shuffle: True

(Refer Slide Time: 09:27)



```
1 "accuracy": total_correct/ len(train_set),
2 "loss": total_loss,
3 },
4 )
5
6 tb.close()
```

epoch: 0 total\_correct: 44640 loss: 1256.9741835147142  
batch\_size: 32 lr: 0.001 shuffle: False  
epoch: 1 total\_correct: 56378 loss: 824.7574753984889  
batch\_size: 32 lr: 0.001 shuffle: False  
epoch: 2 total\_correct: 51618 loss: 728.8565887932837  
batch\_size: 32 lr: 0.001 shuffle: False  
epoch: 3 total\_correct: 52414 loss: 644.8487936995924  
batch\_size: 32 lr: 0.001 shuffle: False  
epoch: 4 total\_correct: 52990 loss: 597.8258985894726

run id: 9  
batch\_size: 64 lr: 0.001 shuffle: True  
epoch: 0 total\_correct: 44238 loss: 657.1984748495682  
batch\_size: 64 lr: 0.001 shuffle: True  
epoch: 1 total\_correct: 56129 loss: 421.2944297641516  
batch\_size: 64 lr: 0.001 shuffle: True  
epoch: 2 total\_correct: 51580 loss: 363.4729793287584  
batch\_size: 64 lr: 0.001 shuffle: True  
epoch: 3 total\_correct: 52215 loss: 328.31485691857444  
batch\_size: 64 lr: 0.001 shuffle: True



(Refer Slide Time: 09:30)

```
epoch: 0 total_correct: 45904 loss: 0.01108417955144  
batch_size: 64 lr: 0.001 shuffle: False  
epoch: 1 total_correct: 48867 loss: 0.00948775470257  
batch_size: 64 lr: 0.001 shuffle: False  
epoch: 2 total_correct: 50662 loss: 0.0082236633764  
batch_size: 64 lr: 0.001 shuffle: False  
epoch: 3 total_correct: 51591 loss: 0.007526504993  
batch_size: 64 lr: 0.001 shuffle: False  
epoch: 4 total_correct: 52240 loss: 0.006524968160255  
  
run id: 11  
batch_size: 128 lr: 0.001 shuffle: True  
epoch: 0 total_correct: 48830 loss: 0.009516088306984  
batch_size: 128 lr: 0.001 shuffle: True  
epoch: 1 total_correct: 47689 loss: 0.0085824963450432  
batch_size: 128 lr: 0.001 shuffle: True  
epoch: 2 total_correct: 50182 loss: 0.00723983834819794  
batch_size: 128 lr: 0.001 shuffle: True  
epoch: 3 total_correct: 51269 loss: 0.006812481264906  
batch_size: 128 lr: 0.001 shuffle: True  
epoch: 4 total_correct: 52021 loss: 0.00565332099438  
  
run id: 12  
batch_size: 128 lr: 0.001 shuffle: False  
epoch: 0 total_correct: 48922 loss: 0.009432644248089  
batch_size: 128 lr: 0.001 shuffle: False  
epoch: 1 total_correct: 47847 loss: 0.0082074113146347  
batch_size: 128 lr: 0.001 shuffle: False  
  
run id: 13  
batch_size: 128 lr: 0.001 shuffle: True  
epoch: 0 total_correct: 48180 loss: 0.0087516983773  
batch_size: 128 lr: 0.001 shuffle: True  
epoch: 1 total_correct: 51798 loss: 0.007434373754263  
batch_size: 128 lr: 0.001 shuffle: True  
epoch: 2 total_correct: 52520 loss: 0.00631632836163044  
batch_size: 128 lr: 0.001 shuffle: True  
epoch: 3 total_correct: 52952 loss: 0.005909071058035  
batch_size: 128 lr: 0.001 shuffle: True
```

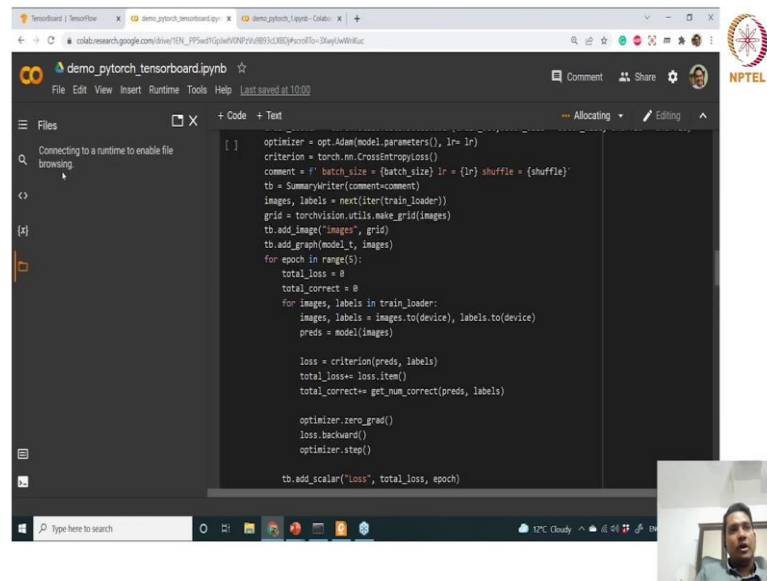
So, you can see that for each run id 1 2 3 . So, we are running actually for 12 times ok for each time how many batch size have been processed with each lr value shuffle? So, this is just entire loop.

(Refer Slide Time: 09:40)

```
run id: 13  
batch_size: 128 lr: 0.001 shuffle: True  
epoch: 0 total_correct: 48180 loss: 0.0087516983773  
batch_size: 128 lr: 0.001 shuffle: True  
epoch: 1 total_correct: 51798 loss: 0.007434373754263  
batch_size: 128 lr: 0.001 shuffle: True  
epoch: 2 total_correct: 52520 loss: 0.00631632836163044  
batch_size: 128 lr: 0.001 shuffle: True  
epoch: 3 total_correct: 52952 loss: 0.005909071058035  
batch_size: 128 lr: 0.001 shuffle: True
```

Now, from this you can see that tracking the performance right. So, which we want to track it is very hard. So, tensor flow will come in handy. So, for tensor flow load. So, load extension tensorboard and to improve to include the logs which is inside the runs directory.

(Refer Slide Time: 10:03)



```
[ ]
optimizer = opt.Adam(model.parameters(), lr=1e-3)
criterion = torch.nn.CrossEntropyLoss()
comment = f' batch_size = {batch_size} lr = {lr} shuffle = {shuffle}'
tb = SummaryWriter(comment=comment)
images, labels = next(iter(train_loader))
grid = torchvision.utils.make_grid(images)
tb.add_image("images", grid)
tb.add_graph(model_t, images)
for epoch in range(5):
    total_loss = 0
    total_correct = 0
    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)
        preds = model(images)

        loss = criterion(preds, labels)
        total_loss += loss.item()
        total_correct += get_num_correct(preds, labels)

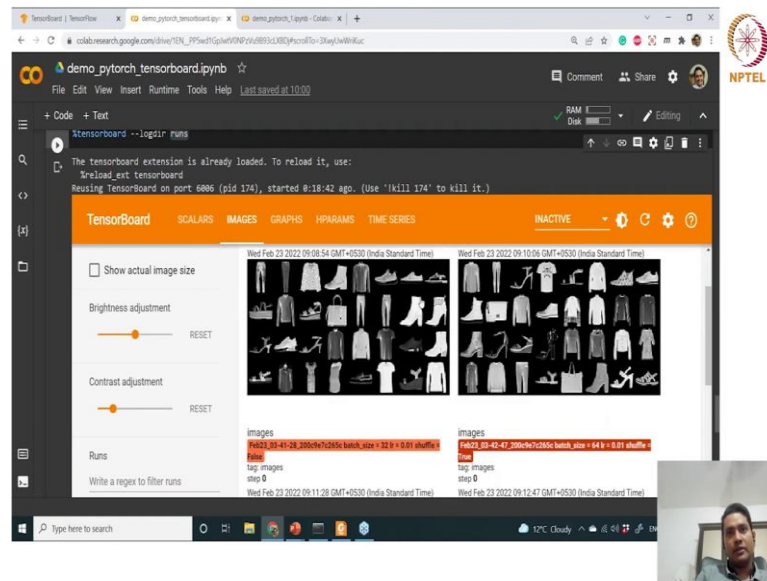
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    tb.add_scalar("loss", total_loss, epoch)
```

So, when you will run this module you will see that one runs directory has been found here ok inside your current directory and the runs directory actually having all the logs which you have written from your tb ok.

So, all these logs for each run will be written and we are actually loading all the logs from runs directory. So, we are log we are. So, this comment will block all the logs from your runs directory or any other directory you are using if you are using other directories you just rename it, but the important thing is that we can also define which run you want to log ok. Here we are loading all the runs ok.

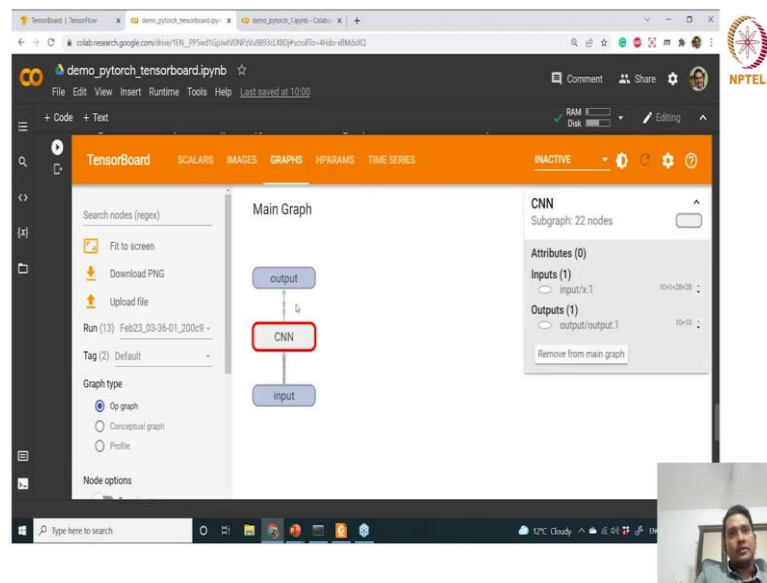
(Refer Slide Time: 11:00)



So, now after running this you will get this tensorboard view here. So, this display will have these images. So, as you can see the images we have loaded. So, this is the a FashionMnist load images that we have loaded from the tabulator. So, just to show you, but from we loaded the images right.

So, when we have written the images the grid we have formed the grid first right with this it will make grid function in the torchvision and then this grid is added to this as images and then the graph is also added as graph. So, we will see the training module graph right.

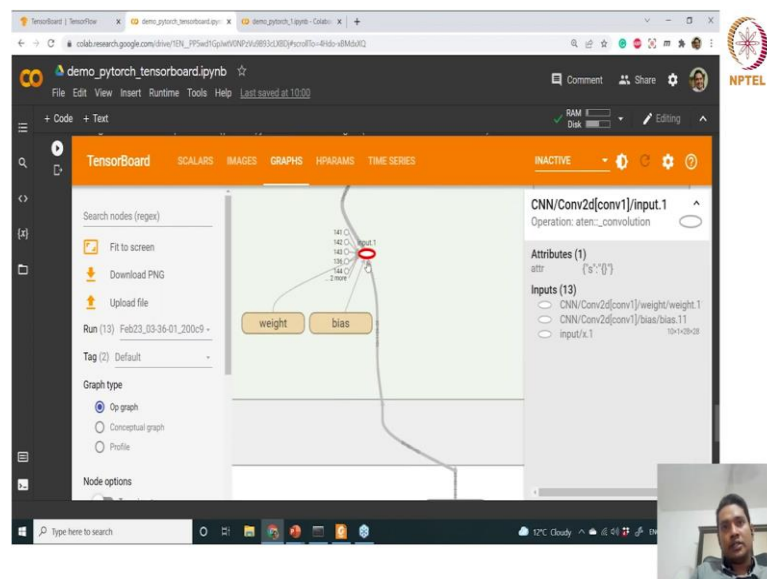
(Refer Slide Time: 11:47)



So, for the graphs you can go to the graphs tab and you can see the graphs are essentially compressed here.

So, the input is the input to your module you can click into any module and you can see what kind of layer it is ok and what is the dimension you can click on to it and it will actually enlarge ok.

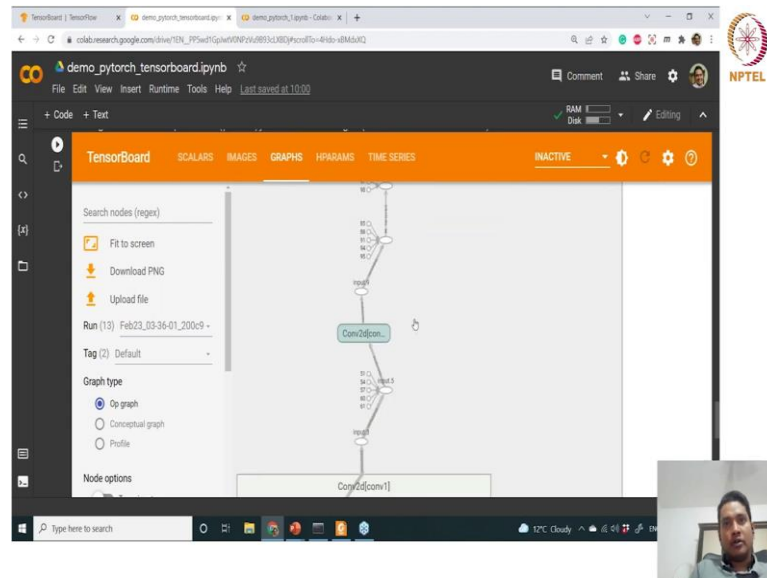
(Refer Slide Time: 12:09)



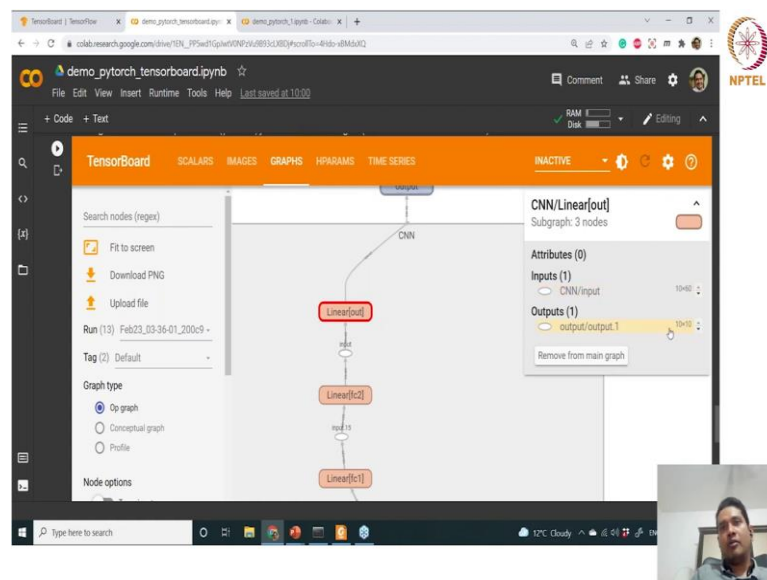
So, basically it will focus on to each layers specifications wise graph ok. So, this is the computational graph that we are talking about. So, in the first class we have talked about

the computational graph. So, basically the input vectors the weights biases now you are if you are doing convolution operation or linear operation which is the fully connected operation that you can do. So, what kind of operation you are doing ok?

(Refer Slide Time: 12:50)

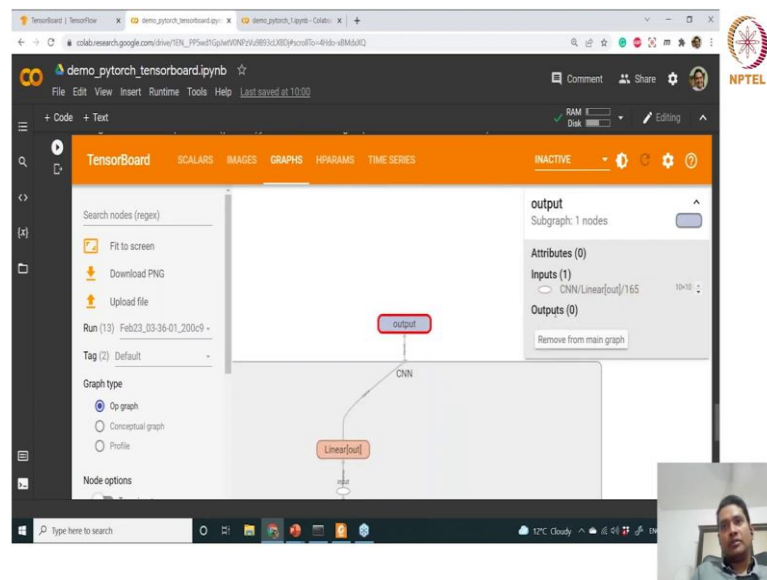


(Refer Slide Time: 12:53)



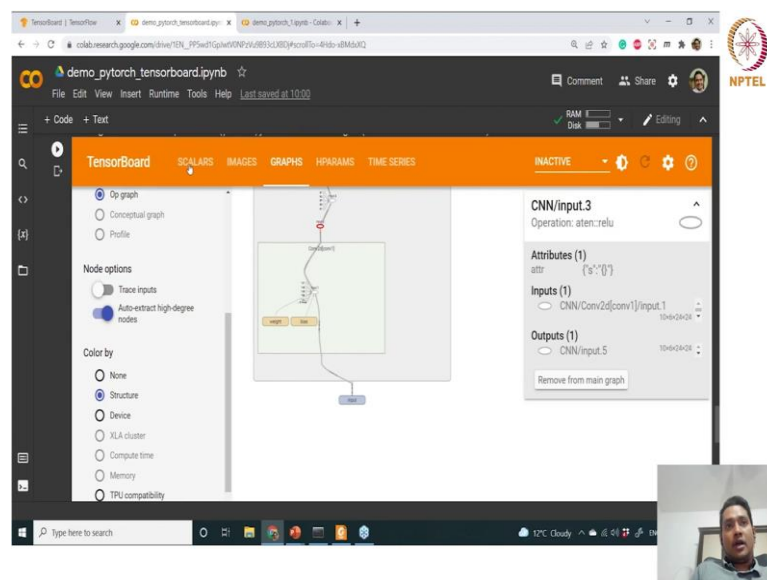
So, you can see for each layer convolution1 then linear fully connected layer fully connected layer2 linear output and also you can see the dimensions ok. So, at the output we are giving 10 and 10 batches ok. So, 10 by 10 is the.

(Refer Slide Time: 13:13)



So, in the output we have linear output ok. So, the entire graph you can see as the graph here.

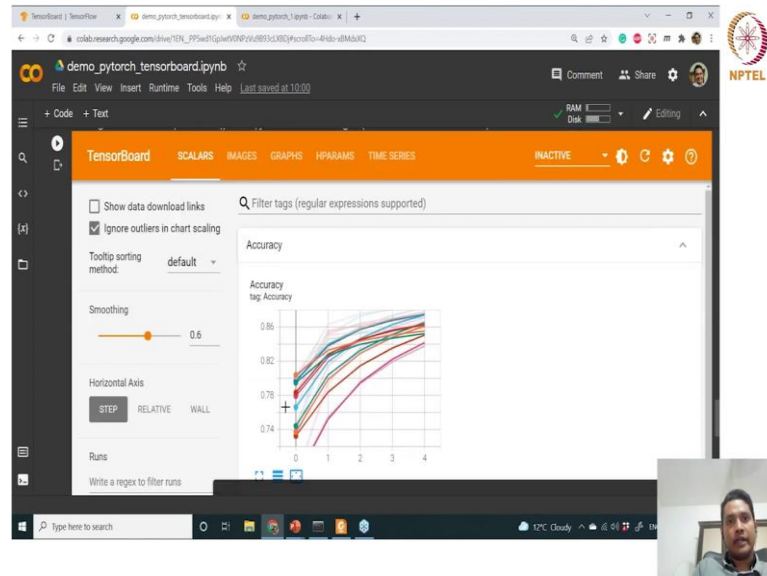
(Refer Slide Time: 13:24)



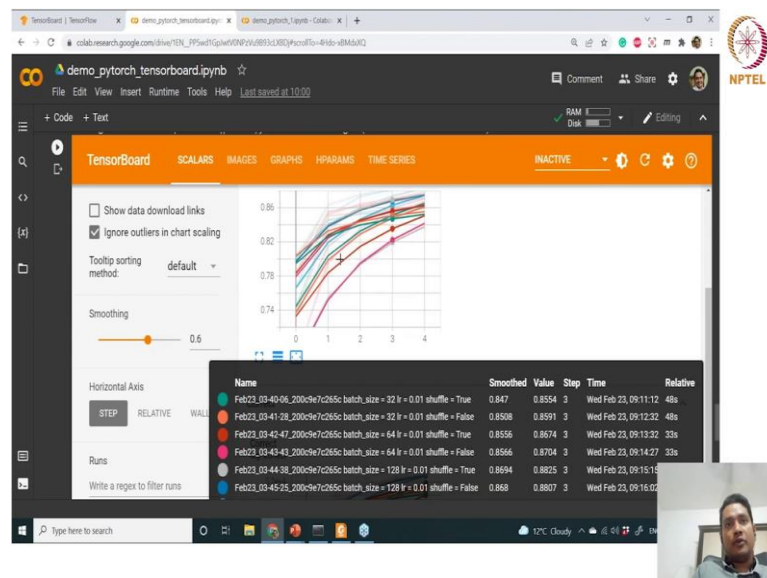
So, once you can analyze from the network definition and the graph actually outcome what is you are getting. So, all the attributes you can see here as well as like whether you are what kind of device I mean in which device it is transferred. So, all these structure you can see now in this scalars. So, as I was mentioning that the graphs are essentially

loss versus your epoch your correctness versus your epoch the accuracy versus epoch.  
So, all these are added inside your scalars tb dot add scalars.

(Refer Slide Time: 14:06)

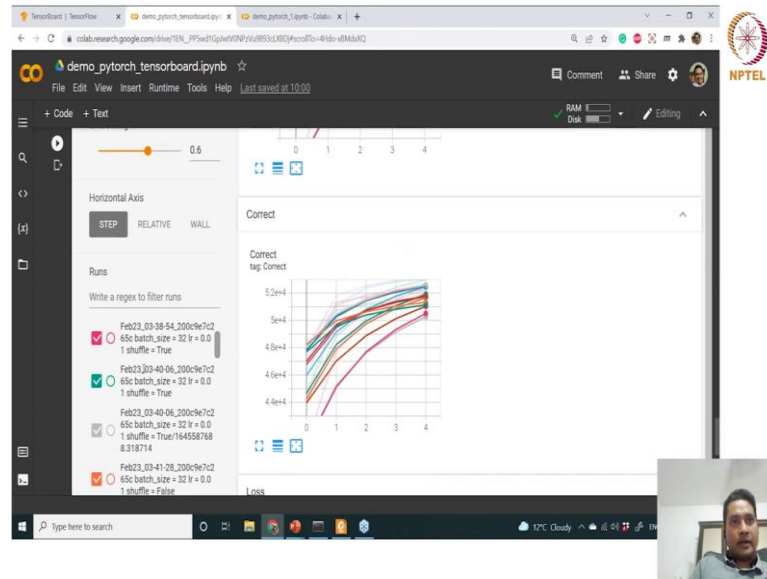


(Refer Slide Time: 14:08)





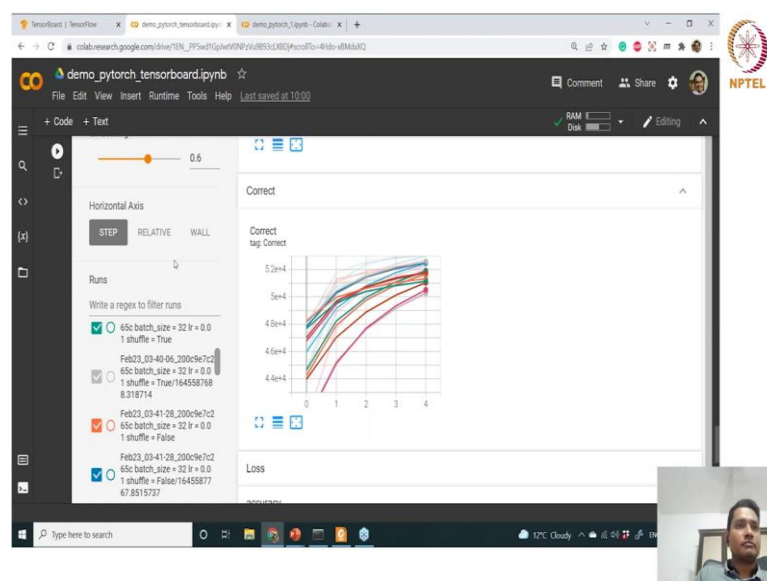
(Refer Slide Time: 14:15)



So, now you can see the accuracy versus epoch graph. So, how for each run ok. So, you can select also this run. So, these are the runs. So, we have renamed it with the date the batch size the shuffle value and error value ok. So, as I was mentioning that for each combination you will get one such profile ok. So, all these profilers we are giving output here ok.

So, that is why we have different logs here you can see ok.

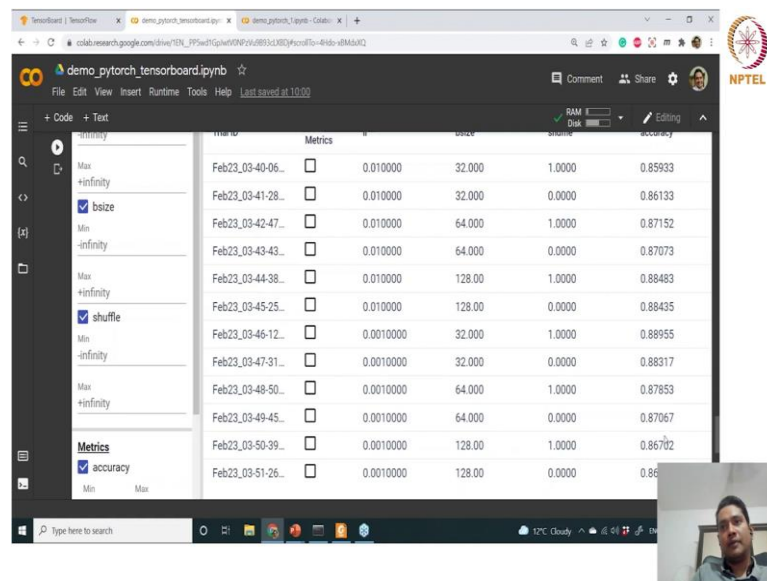
(Refer Slide Time: 14:44)



So, you can also write some like expression to filter the runs also ok. So, this is just some functionality added here now the hyper parameter. So, the important thing I wanted to show here is that the analysis that you are seeing here is not very feasible to or not very feasible to analyze properly right.

So, if you have let us say hundreds of runs using multiple GPUs you actually will get lost. So, that HPARAMS will have the entire parameters a table view ok.

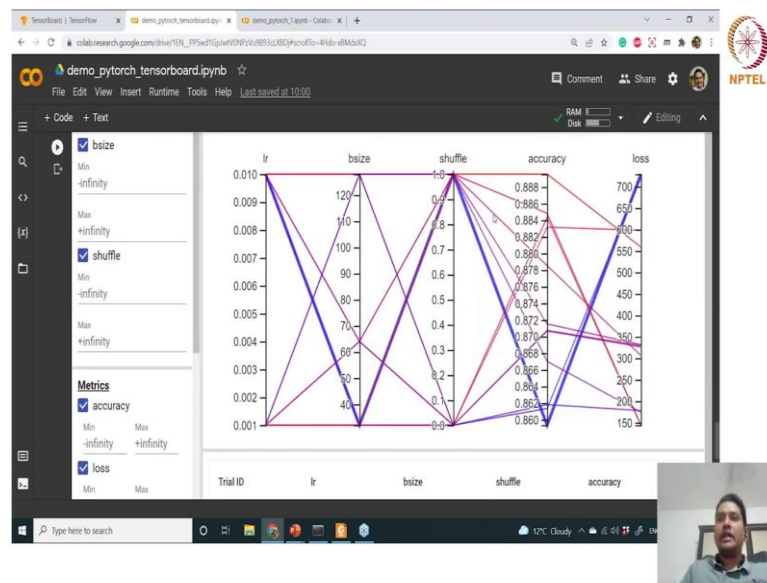
(Refer Slide Time: 15:26)



So, table view will have parallel coordinates view will have which is very important scatter plot matrix view also view will have. So, basically in the table view for each run how many parameters batch size accuracy you can see in the one place ok.

So, this is also how you can analyze or better way to analyze is to see the parallel coordinates view.

(Refer Slide Time: 15:47)

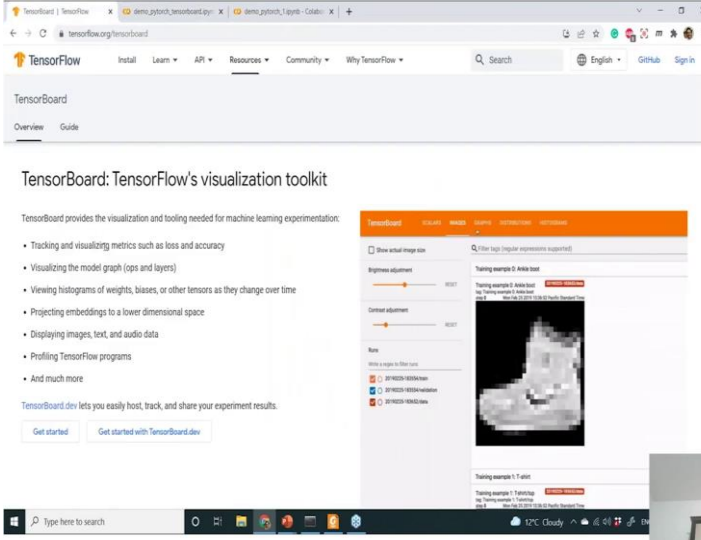


So, now in the parallel coordinates view we can see that loss is essentially increasing and here the accuracy. So, lower accuracy to higher accuracy you are going right. So, this is the highest accuracy you are getting.

Now, for this highest accuracy what is the shuffle value? What is your batch size you can track everything right. What is your batch size and what is the inner value you can track. So, for actually shuffle called True or False ok. So, that those were two definitions for your shuffle and for all the two shuffle have we see that I mean you are getting very less accuracy.

So, for higher accuracy we will go for shuffle equal to two batch size minimum is actually good here you can see the minimum with minimum batch size you are getting good results and the lr is 0.001 for your highest accuracy for this training ok. So, this is just the simple analysis that you can do with tensorboard.

(Refer Slide Time: 17:00)



The screenshot shows the TensorFlow TensorBoard website. The header includes the TensorFlow logo and navigation links: Install, Learn, API, Resources, Community, and Why TensorFlow. A search bar is on the right. The main content area is titled "TensorBoard: TensorFlow's visualization toolkit" and lists various features like tracking metrics, visualizing model graphs, and displaying images. On the right, there's a sidebar with "Training example 1: Audio test" and "Training example 2: T-test". A small video inset in the bottom right corner shows a person speaking.

TensorBoard

Overview Guide

### TensorBoard: TensorFlow's visualization toolkit

TensorBoard provides the visualization and tooling needed for machine learning experimentation:

- Tracking and visualizing metrics such as loss and accuracy
- Visualizing the model graph (ops and layers)
- Viewing histograms of weights, biases, or other tensors as they change over time
- Projecting embeddings to a lower dimensional space
- Displaying images, text, and audio data
- Profiling TensorFlow programs
- And much more

TensorBoard lets you easily host, track, and share your experiment results.

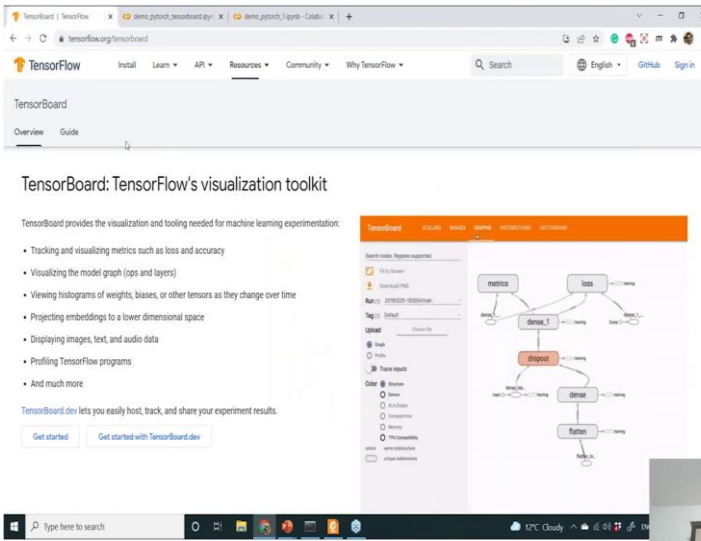
[Get started](#) [Get started with TensorBoard.dev](#)

TensorBoard

Training example 1: Audio test

Training example 2: T-test

(Refer Slide Time: 17:01)



This screenshot shows the same TensorFlow TensorBoard website, but with the "Guide" tab selected. The main content area displays a detailed model graph with nodes like "metrics", "loss", "dense\_1", "dense", and "softmax". The sidebar on the right shows a search bar and a list of metrics. A small video inset in the bottom right corner shows the same person speaking.

TensorBoard

Overview Guide

### TensorBoard: TensorFlow's visualization toolkit

TensorBoard provides the visualization and tooling needed for machine learning experimentation:

- Tracking and visualizing metrics such as loss and accuracy
- Visualizing the model graph (ops and layers)
- Viewing histograms of weights, biases, or other tensors as they change over time
- Projecting embeddings to a lower dimensional space
- Displaying images, text, and audio data
- Profiling TensorFlow programs
- And much more

TensorBoard lets you easily host, track, and share your experiment results.

[Get started](#) [Get started with TensorBoard.dev](#)

TensorBoard

Search metrics, display experiment

metrics

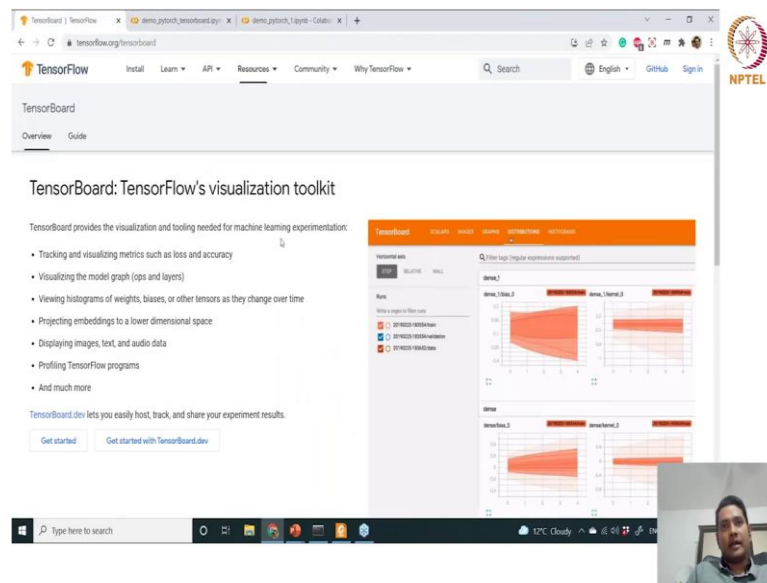
loss

dense\_1

dense

softmax

(Refer Slide Time: 17:04)

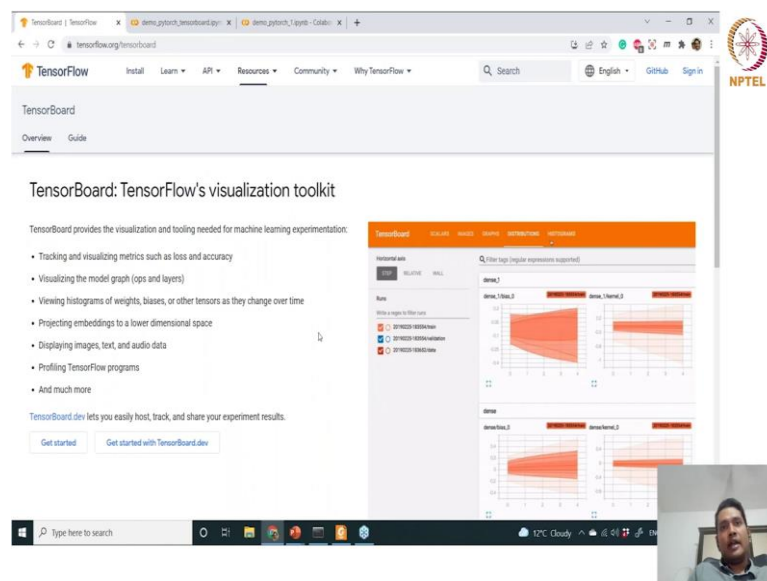


The screenshot shows the TensorFlow TensorBoard website. The page title is "TensorBoard: TensorFlow's visualization toolkit". Below the title, it states "TensorBoard provides the visualization and tooling needed for machine learning experimentation:" followed by a bulleted list of features:

- Tracking and visualizing metrics such as loss and accuracy
- Visualizing the model graph (ops and layers)
- Viewing histograms of weights, biases, or other tensors as they change over time
- Projecting embeddings to a lower dimensional space
- Displaying images, text, and audio data
- Profiling TensorFlow programs
- And much more

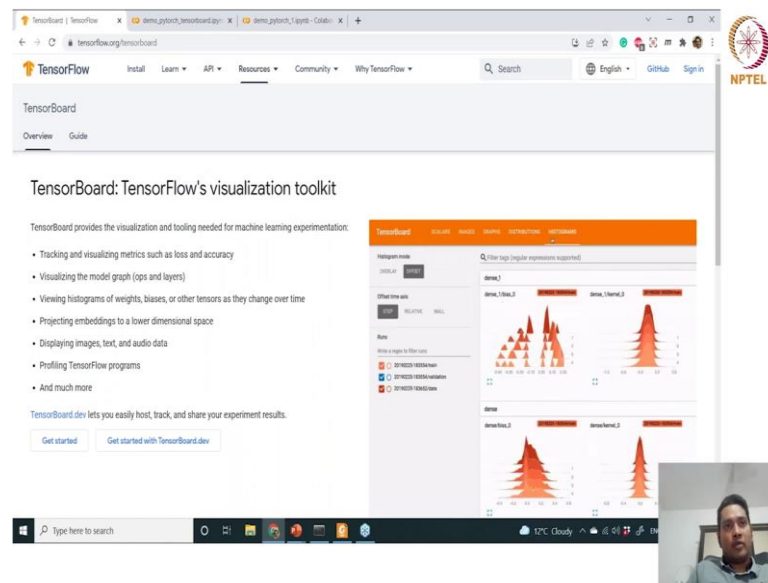
Below the list, it says "TensorBoard.dev lets you easily host, track, and share your experiment results." and provides two buttons: "Get started" and "Get started with TensorBoard.dev". On the right side of the page, there is a preview of the TensorBoard interface showing various plots like "Horizontal axis" and "Runs". In the bottom right corner, there is a small video inset showing a person's face.

(Refer Slide Time: 17:06)

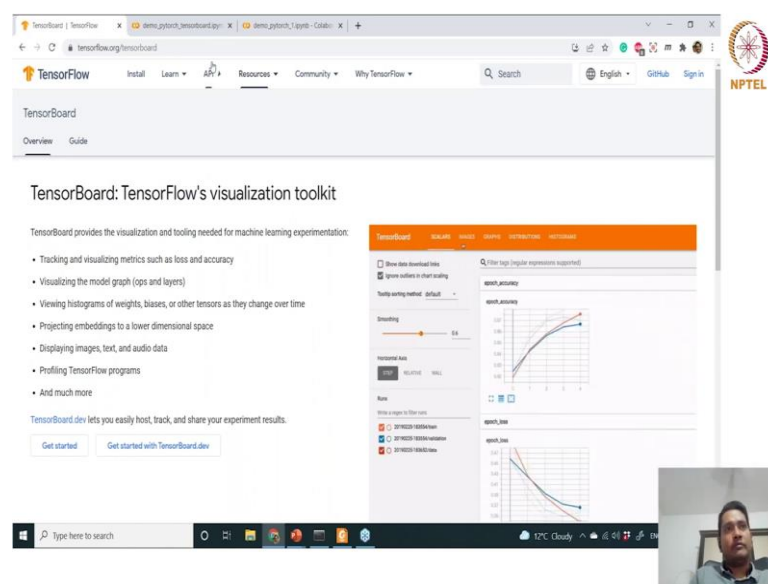


This screenshot is identical to the one above, showing the TensorFlow TensorBoard website. It includes the same text, bulleted list of features, and buttons. The video inset in the bottom right corner shows a different person's face.

(Refer Slide Time: 17:06)



(Refer Slide Time: 17:07)



So, there are many other features that you will like to explore inside this go to TensorFlow TensorBoard and you can explore more about this we will see for a few more features of this tensorboard in the coming classes. In the next session we would like to go towards multi GPU training. So, we would we have seen that if you want to include your training or transfer your training into your device you are actually defining this part right.

So, you are defining the device which is the device here. So, you are actually transferring the model to your device and transferring the data to your device now we want to run. So, this device is only a single GPU here ok. Now we will talk about how we can scale our training for multiple GPUs ok. So, this is this is very interesting this is very simple as well in PyTorch ok.

(Refer Slide Time: 18:04)

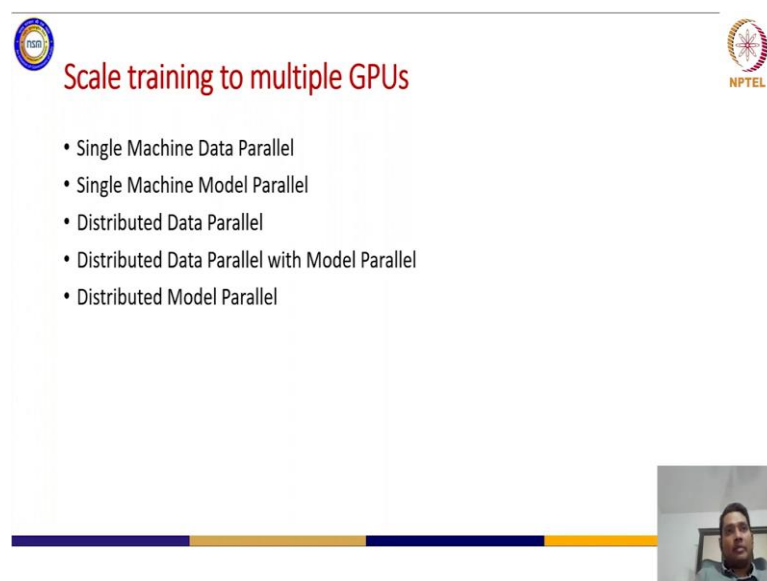
The slide is titled "Scale training to multiple GPUs" in red text. It features two bullet points: "Data Parallel – Data distributed across devices" and "Model Parallel – Model distributed across devices". The slide includes logos for IITM (Indian Institute of Technology Madras) and NPTEL (National Programme on Technology Enhanced Learning). A small video inset of a speaker is visible in the bottom right corner of the slide area.

So, we have data parallel approach we have model parallel. So, data parallel approach is essentially you want to keep the model ok replicated into let us say several GPUs, but data you want to set it right and model parallel is the entire model you want to distribute across devices now they are. So, as mean suggests you have actually two such libraries available in PyTorch two data parallel and model parallel plus data parallel ok.

So, when you are doing model parallel you can do data parallel as well right because since you are distributing the model you can distribute the data also ok. So, this is just the one wrapper up to your data parallel.



(Refer Slide Time: 18:58)



The slide is titled "Scale training to multiple GPUs" in red text. It features a list of five bullet points: "Single Machine Data Parallel", "Single Machine Model Parallel", "Distributed Data Parallel", "Distributed Data Parallel with Model Parallel", and "Distributed Model Parallel". The slide includes logos for "IITM" and "NPTEL" in the top corners. A small video inset in the bottom right corner shows a man speaking.

- Single Machine Data Parallel
- Single Machine Model Parallel
- Distributed Data Parallel
- Distributed Data Parallel with Model Parallel
- Distributed Model Parallel

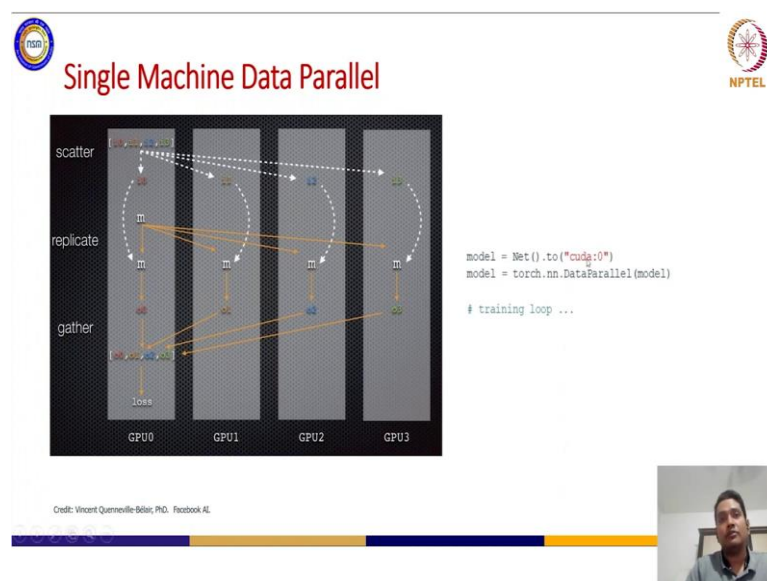
Now, there may be different combinations of configurations that you might have put right. So, you can have single machine data parallel you can have single machine model parallel because in multiple GPUs you want to pipeline several models parallel execution ok several models parallel execution one single model will be actually executed parallelly.

You can pipeline several models training in the same pattern because we will show you like how this is happening inside multiple device single model parallelly and that way you will understand like how it will be you know pipelined. So, this is very simple, but you just if you see the figure it will be very easy to understand distributed data parallel again data parallel in the distributed configuration set up.

So, where you have multiple nodes. So, single node multiple GPUs you can have or multiple node multiple GPUs you can have; that means, multiple node and each node may have multiple GPUs that is how the distributed data parallel set up or distributed set up rather defines.

Now, distributed data parallel with model parallel also we can write. Now distributed model parallel will. So, as I was mentioning that distributed essentially is one wrapper up on the data parallel. So, basically you can have data parallel in the distributed both the data parallel and model parallel and single model parallel.

(Refer Slide Time: 20:40)



So, now we talk about the single machine data parallel.


So, now you have multiple GPUs available inside your single node ok. So, single node here means single machine ok now how your. So, now, we are talking about the data parallel; that means, each GPU will be operating on multiple sets I mean different sets of data right. So, that is why the i 0, i 1, i 2, i 3. So, all these data that you are seeing. So, basically your data is coming in batches.

Now, depending on the number of GPUs are available you can scatter the data to your multiple GPUs then you can replicate your model to several GPUs, you can clean all this data parallel you will get some loss at the end of the. So, remember the pipeline that we have talked about at the end of the training we will get the loss and all these losses you will gather in the master node master GPU node or let us say GPU0 ok if you define at GPU0. So, GPU0 node and you will calculate the mean of these and then actually we will compute the final nodes and then again this process will be repeated right.


So, this loop will go on depending on how many data you have and how many epochs doing this in. So, you have seen the training pipeline this modification only will do your data parallel activation. So, basically now we are transferring the model into let us say cuda 0 ok. So, we have let us say cuda 0 and 1, 2, 3, 4 ok or 5, 6, 7. So, total 8 GPUs you might have or 4 GPUs in this case.

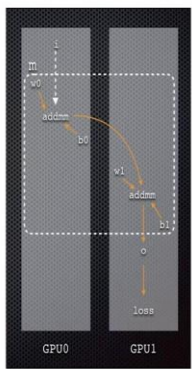
So, cuda 0. So, first model you are trying to I mean which model you are trying to replicate that model you are transferring to only single GPU first and then you are applying data parallel model and when you apply this data parallel nn dot data parallel a function automatically your model will be replicated from the GPU0 to all the other GPU GPUs and depending on the number of GPUs available and the batches you have fetching it will actually segregate or scatter the data into your GPUs right. So, this is very simple.

(Refer Slide Time: 23:00)



## Single Machine Model Parallel





Credit: Vincent Quenneville-Bélair, PhD, Facebook AI

```

class Net(torch.nn.Module):

    def __init__(self, *gpus): super(Net).__init__(self)

        self.gpu0 = torch.device(gpus[0])
        self.gpu1 = torch.device(gpus[1])


        self.sub_net1 = torch.nn.Linear(10, 10).to(self.gpu0)
        self.sub_net2 = torch.nn.Linear(10, 5).to(self.gpu1)

    def forward(self, x):
        y = self.sub_net1(x.to(self.gpu0))
        z = self.sub_net2(y.to(self.gpu1)) # blocking
        return z

model = Net("cuda:0", "cuda:1")

# training loop...

```




Now, single machine multiple model parallel. So, sorry single machine single model parallel ok we are not here talking about multiple models ok. So, just stay with single model now. So, we have several stages of pipeline I mean several stages of the training pipeline we have seen that right. So, what are the pipelines you want to let us say execute in which GPU can define that and depending on that your data synchronization will happen and the in the end of all the sequences are happening then we can actually compute the loss.

Now, when this sequence is happening at that point of time you can actually feed through a different layer of different model into the first GPU while it is doing the second sequence of the first model ok. So, this is how you can pipeline it, but basically this is the code what will parallelize the model execution into different GPUs that you might have.


So, here let us say we have two gpus. So, we are gathering those gpu ids into gpu0 and gpu1 and then we are actually transferring these models defining this layers. So, this is just the simple definition of the layer that we want to execute in which gpu. So, transferring the sequences of the training into the gpus ok then we are doing the forward pass now which in the a net will be executed in which device you can just transfer it into that gpu and that is it. So, one you once you have defined like this model is essentially running in let us say two gpus ok.

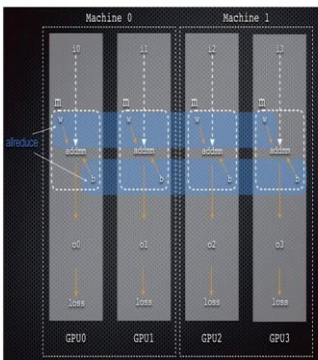
So, this is very simple. So, this is these are the update; upgradation of the entire training pipeline that you have seen that needs to do that to enable this model parallels you can also distribute data parallel. So, basically now we are talking about distributed data parallel ok.

(Refer Slide Time: 25:13)



## Distributed Data Parallel





```
def one_machine(machine_rank, world_size,
                backend): torch.distributed.init_process_group(
    backend, rank=machine_rank, world_size=world_size
)

gpus = {
    0: [0, 1],
    1: [2, 3],
}[machine_rank] # or one gpu per process to avoid GIL

model = Net().to(gpus[0]) # default to first gpu on
machine
model = torch.nn.parallel.DDP(model, device_ids=gpus)

# training loop...

for machine_rank in range(world_size):
    torch.multiprocessing.spawn(
        one_machine, args=(world_size, backend),
        nprocs=world_size, join=True # blocking
    )
```

Credit: Vincent Quenelle-Bélat, PhD, Facebook AI

So; that means, you have several nodes with multiple GPUs right. So, machine 0 has let us say two GPUs machine 1 has 2 GPUs GPU 2 and GPU 3.


Now, what we are doing here? So, in the data parallel we have seen that the model will be actually replicated into several GPUs and all the data will be scattered and feed to the GPUs and losses will be computed. So, this data parallel approach is again very simple to parallelize because once you have defined multiple gpus in multiple nodes. So, let us say we have two nodes here 0 and 1 and with 0 and 1 and 2 and 3 which is the definition or ids of the gpus for each machine we are creating this list here ok.

So, one. So, basically you need to define the rank also because the number of rank will define how many processes will be created or the replicas will be created. So, basically to avoid GIL. So, which is the global interrupt lock because actually you need to synchronize in each round of loss computation ok. So, that is why how many number of gpus you have those many number of ranks definition is the tumble.


So, then you can define the model into the gpu 0 because from the gpu 0 actually it will be replicated to other gpus when you will define this nn.parallel.DDP. So, DDP is essentially the data distributed data parallel library ok. So, then we can actually use this model and this devices ok.

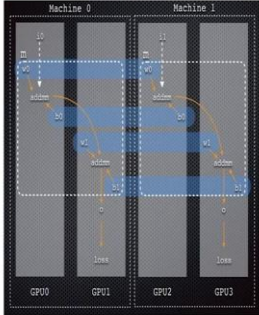
So, this is just the same as data parallel you can see one wrapper up of data parallel approach ok and then you have training loop and for each machine rank in the world size. So, world size is essentially the total number of processes in all the machines and rank is the total number of processes in each machine ok and depending on that it will actually create multiple threads and the. So, this is just to for conjoin your processes. But ultimately your code for training will differ this much only ok.

(Refer Slide Time: 27:59)



## Distributed Data Parallel with Model Parallel





Credit: Vincent Quenelle-Bellier, PhD. Facebook AI


```
def one_machine(machine_rank, world_size, backend):
    torch.distributed.init_process_group(
        backend, rank=machine_rank, world_size=world_size
    )

    gpus = {0: [0, 1],
            1: [2, 3],}[machine_rank]

    model = Net(gpus)
    model = torch.nn.parallel.DDP(model)

    # training loop...

for machine_rank in range(world_size):
    torch.multiprocessing.spawn(
        one_machine, args=(world_size, backend),
        nprocs=world_size, join=True
    )
```



So, next which is the distributed data parallel with model parallel ok. So, again multiple nodes we have. So, this multiple nodes having multiple GPUs we have seen and if you remember the model parallel in a single GPU ok single machine multiple gpu.

So, you have defined which layer actually. So, let us say sub net1 and sub net2 are the 2 main sequences in your training ok in your training module and let us say they are simply fully connected layer ok for simplicity we are taking fully connected layer. Now which layer will go to which device that we have defined. So, basically gpu0 and 1 we have defined, but we have made available these sequences to those gpus.

Because you want to parallelize the model execution to the gpus and also when you are doing the forward pass you need to also get the data available for that because you can see the output of the this layer is going to the input of this layer. So, if you do not give y as the input to the second layer which is the gpu1 because your next layer is they being executed in the gpu1. So, this pipeline you need to set up.

So, this is what I was talking about when you are trying to replicate sorry when you are trying to parallelize the model execution in accordance with defining which layer will go to which gpu which data also will go to which gpu it will you need to define it properly otherwise you will have wrong loss computation ok.

Now, we are talking about the data parallel as well as model parallel ok. So, in multiple nodes you have multiple GPUs in one node you are trying I mean in all the nodes I mean basically you are trying to parallelize the model execution with parallel datas ok. So, that is why again you can see the data parallel is always there with distributed model parallel ok.

So, this is the inherent functionality that distributed data parallel we will use. Now when you are trying to define this because let us say you have gpus in 0 and 1 node you have in 0 node 0 and 1 and 1 node 2 and 3 with these machine rank. Now we have I mean converted a model into this parallel and this is the training loop which will run ok.

Now, once you have you are working with DDP ok not the data parallel right. So, again just to make sure that you remember this part. So, this is actually we are not using the distributed data parallel ok. So, this is just we are transferring which model sequence will go to which gpu ok. So, we are not using any defined library ok.

Now, here once we are using DDP you when you are. So, that time we needed to define which sequence will go and how the output and input will make the connection between different gpus outputs and inputs, but here since you are using DDP

`torch.nn.parallel.DDP` you do not need to define that we just it will just take care of all the data parallel scatter.

So, basically what it will do? It will scatter the data according to different replicas of the model and different replicas of the model will have different sequences are repeated to different GPUs inside one node ok. And then how many process these are the standard process falling ok. So, this is this will be used. So, this is how you can just scale your training entire pipeline that you have seen so, far with the use of these two libraries one is the data parallel library and one is distributed data parallel library in PyTorch.

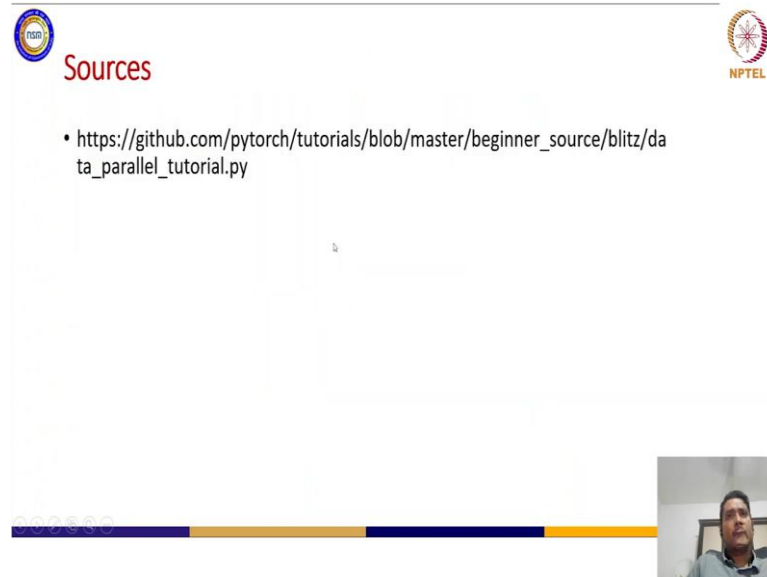
There are also other libraries and other packages available you can use. So, once you know actually the basics of these two libraries there is also RPC which is remote procedure call from from PyTorch. So, basically we will not have the bandwidth to discuss that package, but we can go to [pytorch.org](https://pytorch.org) to get details of that package as well.

But apart from the standard packages that is available with PyTorch. So, all these are developed by Facebook and also the development is continuously getting updated there might be few bugs in the previous versions next version will come up with the fixes. So, all these things you will see, but there is a strong community available for that and you can actually also contribute if you want for developing these libraries.

So, this is completely open source project and apart from that I was talking about there is also some perpetual libraries that is using wrapped PyTorch modules to abstract more of these tasks even in fancier ways. So, that also you can see in many other libraries.



(Refer Slide Time: 34:32)



The slide is titled "Sources" in red text. It features a bulleted list with a single item: [https://github.com/pytorch/tutorials/blob/master/beginner\\_source/blitz/data\\_parallel\\_tutorial.py](https://github.com/pytorch/tutorials/blob/master/beginner_source/blitz/data_parallel_tutorial.py). The slide includes logos for IIT Bombay (top left) and NPTEL (top right). At the bottom right, there is a small video feed showing a man speaking. The bottom of the slide has a navigation bar with several icons and a blue and yellow color scheme.

So, few of them maybe we will discuss in the next class, but this class we will conclude here and for the code you can go to this link for getting access to one very very intuitive implementation of that data parallel. So, basically we have seen how to actually tweak your entire frame with just few lines upward to make it available to multiple gpus. So, so one such example is there in this for which is available in this link.

Thank you for today we will now go towards the questions essentially.